Tridex
March 27, 2018

SQL Enhancements Delivered with Db2 12 for z/OS

Robert Catterall, IBM
rfcatter@us.ibm.com

Agenda

- Temporal enhancements
- Enhancements related to transparent archiving
- Advanced triggers
- SQL PL enhancements
- The new and improved MERGE statement
- SQL pagination
- Piece-wise DELETE
- Array and global variable enhancements
- “Real” Unicode columns in EBCDIC tables
- “Local” ODBC driver enhancements
- Optional correlation clause for table expressions
- New built-in functions
- Native REST interface

System-time temporal: tracking data changes

- Track “what” and “who” of data changes via two GENERATED ALWAYS AS columns
  - What: GENERATED ALWAYS AS (DATA CHANGE OPERATION)
    - Possible values: ‘I’ for INSERT, ‘U’ for UPDATE, ‘D’ for DELETE
  - Who: GENERATED ALWAYS AS (SESSION_USER)
    - Or CURRENT SQLID, or CURRENT CLIENT_WRKSTNNAME, or CURRENT CLIENT_USERID, …
  - Also available in Db2 11 environment (APAR PM99683)
- Example:

Table **BANK_ACCOUNT**

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END

GENERATED ALWAYS AS (SESSION_USER)

GENERATED ALWAYS AS (DATA CHANGE OPERATION)

Temporal data-change tracking in action

1. User JOE inserts row for ACCOUNT_ID 56789

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END
56789	1234.56	JOE	I	2017-01-19	9999-12-30

BANK_ACCOUNT

2. User DON updates the row

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END
56789	88.77	DON	U	2017-01-21	9999-12-30

BANK_ACCOUNT

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END
56789	1234.56	JOE	I	2017-01-19	2017-01-21

BANK_ACCOUNT_**HIST**

3. User LAURA deletes the row

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END

BANK_ACCOUNT

ACCOUNT_ID	BALANCE	USER	OP_CODE	SYS_START	SYS_END
56789	1234.56	JOE	I	2017-01-19	2017-01-21
56789	88.77	DON	U	2017-01-21	2017-02-15
56789	88.77	LAURA	D	2017-02-15	2017-02-15

BANK_ACCOUNT_**HIST**

"Extra" history row for DELETE: specify ON DELETE ADD EXTRA ROW on ALTER statement that activates versioning

Business-time temporal: support for inclusive/inclusive

- Before Db2 12: business time start/end dates had inclusive/exclusive meaning

First day on which coverage of 20000 will be in effect

POLICY_ID	COVERAGE	BUS_START	BUS_END
A123	20000	2016-01-01	2016-07-01
A123	30000	2016-07-01	2016-09-01
A123	20000	2016-09-01	2018-01-01

First day on which coverage of 20000 will **NOT** be in effect

- Some users said, ”That’s not how we think of business time dates,” so Db2 12 provides an inclusive/inclusive option for business time dates (V12R1M500)

POLICY_ID	COVERAGE	BUS_START	BUS_END
A123	20000	2016-01-01	2016-06-30
A123	30000	2016-07-01	2016-08-31
A123	20000	2016-09-01	2017-12-31

Last day on which coverage of 20000 **will still be in effect**

More on business-time inclusive/inclusive option

If not specified, defaults to EXCLUSIVE

- To get it: use new INCLUSIVE (or EXCLUSIVE) keyword in DDL

```
CREATE TABLE ...PERIOD BUSINESS_TIME (BUS_START, BUS_END INCLUSIVE)
```

- CHECKCONDITION column of SYSIBM.SYSCHECKS shows option in use:
  - For inclusive/inclusive, you'll see "BUS_END" >= "BUS_START"
  - For inclusive/exclusive you'll see "BUS_END" > "BUS_START"
- Note that form of a temporal update is a little different when INCLUSIVE/INCLUSIVE is used, versus INCLUSIVE/EXCLUSIVE:
  - For inclusive/inclusive, specify BETWEEN <value1> AND <value2>
  - For inclusive/exclusive, specify FROM <value1> TO <value2>

System-time: temporal logical transactions

- Prior to Db2 12: multiple changes to a row in a single unit of work will not be reflected in system-time history table

INSERT at timestamp 2017-01-19-12.34.00 – no commit

Base table BANK_ACCOUNT

ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END
56789	1234.56	1.00	I	2017-01-19-12.34.00	9999-12-30-....

Same transaction: update at timestamp 2017-01-19-12.35.17 – then commit

ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END
56789	987.12	1.00	U	2017-01-19-12.35.17	9999-12-30-....

History table BANK_ACCOUNT_HIST

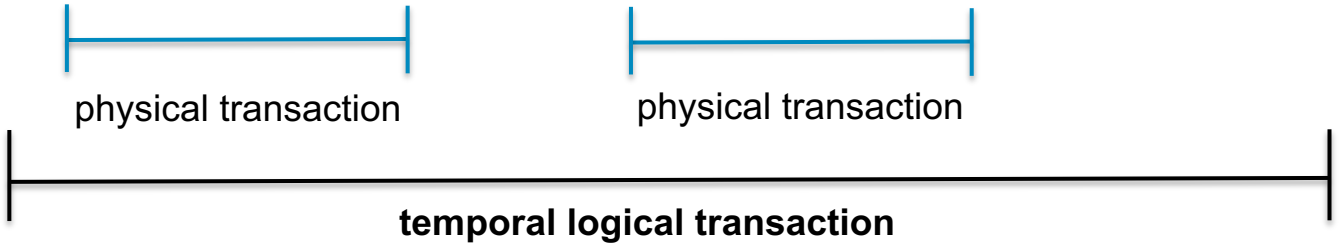
ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END

No “before” image of updated row in history table, because UPDATE occurred in same unit of work as INSERT

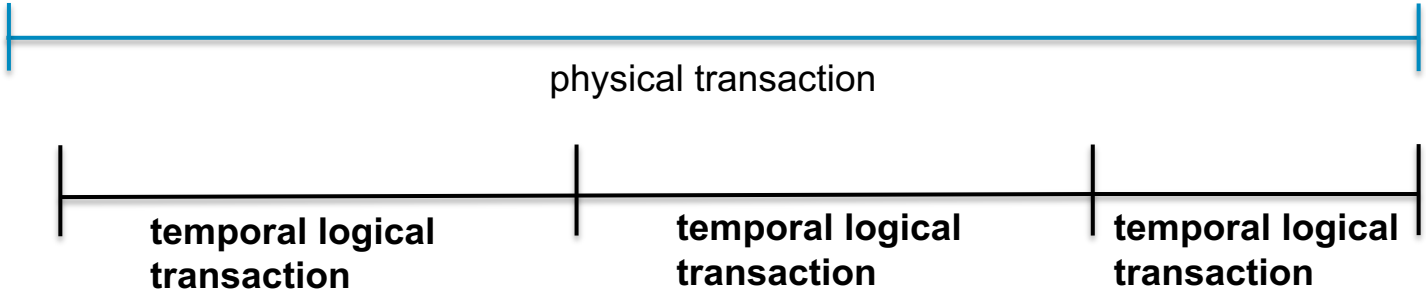
Db2 12: temporal logical transactions (TLTs) enable new behavior

(V12R1M500)

- Now possible to de-couple physical transactions and creation of history table records
- Scenario: multiple physical transactions, one temporal logical transaction
  - Maybe you want changes made by multiple UOWs of a batch job to have same “start” time



- Scenario: multiple TLTs in one physical transaction
  - Referring to previous slide, maybe you want effect of UPDATE reflected in history table



Temporal logical transactions: two new global variables

- TEMPORAL_LOGICAL_TRANSACTION_TIME
  - Provides “start” timestamp value for row versioning in base table for a TLT
  - Default value is NULL (in which case TLT functionality is not in effect)
- TEMPORAL_LOGICAL_TRANSACTIONS
  - When set to 1, there can be multiple TLTs within a single physical transaction
    - Note: multiple changes to a row in one TLT will result in one history table row insert
  - Default value of 0 means multiple TLTs in one physical transaction not allowed

TLT functionality in action

This action means, "Allow multiple TLTs in one physical transaction"

```
SET TEMPORAL_LOGICAL_TRANSACTIONS=1;
```

"Start" timestamp for base table row versioning for TLT (can be CURRENT TIMESTAMP)

```
SET TEMPORAL_LOGICAL_TRANSACTION_TIME = '2017-01-19-14.00.00';
```

INSERT new account at 2017-01-19-12.34.00 – no commit

Table BANK_ACCOUNT

ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END
56789	1234.56	1.00	I	2017-01-19-14.00.00	9999-12-30-23.59.99....

"Start" timestamp for row versioning for another TLT

```
SET TEMPORAL_LOGICAL_TRANSACTION_TIME='2017-01-19-14.15.00';
```

Same physical transaction: update at 2017-01-19-12.35.17, then commit

ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END
56789	987.12	1.00	U	2017-01-19-14.15.00	9999-12-30-23.59.99....

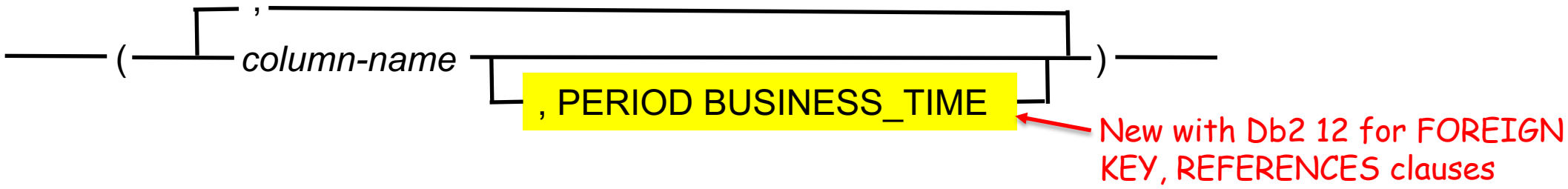
History table BANK_ACCOUNT_HIST

ACCOUNT_ID	BALANCE	FEE	OP.	SYS_START	SYS_END
56789	1234.56	1.00	I	2017-01-19-14.00.00	2017-01-19-14.15.00

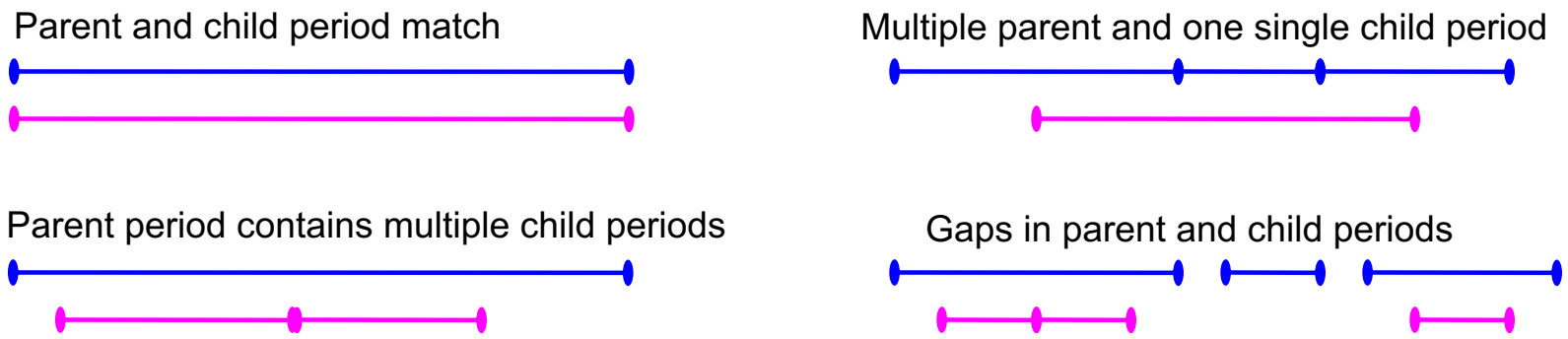
History table now populated because INSERT and UPDATE were in different TLTs

Business-time temporal RI enhancement (V12R1M500)

- New syntax for FOREIGN KEY and associated REFERENCES clauses of ALTER TABLE and CREATE TABLE:



- What this means: Db2 will ensure no “business-time temporal orphans”
  - In other words, child row business time period(s) will always be covered by parent row period(s)



More on business-time temporal RI

- Parent key must have unique index with BUSINESS_TIME WITHOUT OVERLAPS
- Foreign key must have index with BUSINESS_TIME WITH OVERLAPS
- Self-referencing constraints not supported (i.e., parent and child tables must be different)
- The ON DELETE RESTRICT rule is required
- At this time, no support for temporal UPDATE/DELETE (i.e., UPDATE or DELETE with FOR PORTION OF BUSINESS_TIME) for parent table when business-time RI is in effect – even if no child rows exist

Business-time temporal RI: DDL example

Parent table and index

```
CREATE TABLE CAR_POLICY
(POLICY_ID INTEGER NOT NULL, COVERAGE INTEGER NOT NULL,
BUS_START DATE NOT NULL, BUS_END DATE NOT NULL,
PERIOD BUSINESS_TIME (BUS_START, BUS_END).
PRIMARY KEY (POLICY_ID, BUSINESS_TIME WITHOUT OVERLAPS)) IN...;

CREATE UNIQUE INDEX IX_CAR ON CAR_POLICY (POLICY_ID, BUSINESS_TIME WITHOUT
OVERLAPS);
```

Child table and index

```
CREATE TABLE LINE_ITEM (ITEMID INTEGER NOT NULL,
POLICY_ID, BUS_START DATE NOT NULL, BUS_END DATE NOT NULL,
FOREIGN KEY (POLICY_ID, PERIOD BUSINESS_TIME)
REFERENCES CAR_POLICY (POLICY_ID, PERIOD BUSINESS_TIME) ON DELETE RESTRICT,
PERIOD BUSINESS_TIME (BUS_START, BUS_END)) IN ...;

CREATE INDEX IX_LINE ON LINE_ITEM (POLICY_ID, BUSINESS_TIME WITH OVERLAPS);
```

New Db2 12 syntax

Db2-managed archiving (aka transparent archiving) – update

- Transparent archiving introduced with Db2 11
  - Enables concentration of newer, more frequently retrieved rows in base table, with “older and colder” rows stored in associated archive table – programs see single logical table
- Db2 12:
  - New ZPARM, MOVE_TO_ARCHIVE_DEFAULT, specifies default value for MOVE_TO_ARCHIVE global variable
    - Retrofitted to Db2 11 via APAR PI56767
  - Range-partitioned table can be partitioned on ROW CHANGE TIMESTAMP column
    - Enables Db2 transparent archiving to be effectively paired with high-performance storage saver feature of Db2 Analytics Accelerator
    - Retrofitted to Db2 11 via APAR PI63830
  - Db2 12 optimizer: significantly better performance for queries involving UNION ALL
    - Transparent archiving can transform a query to include a UNION ALL of base table and archive table
    - Db2 12 enhanced UNION ALL performance can also be beneficial for queries targeting system-time temporal tables (Db2 can automatically generate UNION ALL queries for these tables, too)

(SQL Procedure Language)

SQL PL enhancements – triggers

- What were called “triggers” before Db2 12 are called “basic triggers” in a Db2 12 environment
- Db2 12 introduced “advanced triggers,” which deliver multiple benefits, including:
  - Body of advanced trigger can include SQL PL (e.g., logic flow control statements such as IF, ITERATE, LOOP, and WHILE)
    - Enables easier creation of more highly-functional triggers (previously, advanced functionality relied on trigger calling a stored procedure)
  - Provides compatibility with Db2 for Linux/UNIX/Windows
  - Multiple versions of a given trigger can be defined and maintained (ADD VERSION and ACTIVATE VERSION are options for ALTER TRIGGER)
  - And, new CREATE TRIGGER capabilities solve a problem that can be encountered when several triggers have been defined for a table  (more on next slide)
  - Available with function level V12R1M500

Changing a trigger while preserving "firing order"

- The problem (before advanced triggers):
  - If multiple triggers defined on a table have the same activation time (e.g., AFTER) and are "fired" (i.e., activated) by a given SQL operation, the order of firing depends on the order in which they were created
  - Changing a trigger requires drop/re-create, and that could change firing order unless ALL relevant triggers are dropped/re-created in desired order
- Advanced triggers – problem solved!
  - Now, 3 options for changing existing trigger without affecting firing order:

New syntax — ID of version being changed

```
CREATE OR REPLACE TRIGGER trigger-name VERSION version-ID…
```

New syntax — ID of version being changed

```
ALTER TRIGGER trigger-name REPLACE VERSION version-ID…
```

New syntax — New version ID

```
ALTER TRIGGER trigger-name ADD VERSION version-ID…
ALTER TRIGGER trigger-name ACTIVATE VERSION version-ID…
```

Example of an advanced trigger

As with other SQL PL routines, debug with Data Studio

These are among new CREATE TRIGGER options

SQL PL logic flow control statements

Trigger body contains this logic:

If class end time is null, value is set to 1 hour after start of class; otherwise, if class ends after 9pm then an error is returned

```
CREATE TRIGGER MYTRIG01
BEFORE INSERT ON MYTAB
REFERENCING NEW AS N
FOR EACH ROW
ALLOW DEBUG MODE
QUALIFIER ADMF001
WHEN(N.ending IS NULL OR n.ending > '21:00')
L1: BEGIN ATOMIC
        IF (N.ending IS NULL) THEN
            SET N.ending = N.starting + 1 HOUR;
        END IF;
        IF (N.ending > '21:00') THEN
            SIGNAL SQLSTATE '80000'
            SET MESSAGE_TEXT =
                    'Class ending time is beyond 9 pm';
        END IF;
        SET GLOBAL_VAR = NEW.C1;
END L1#
```

With advanced trigger, SET is not restricted to transition variables – it can also be used with global variables and SQL variables (latter refers to variables declared in body of trigger)

SQL PL enhancements – PREPARE in SQL function

(available with function level V12R1M500)

- Compiled SQL scalar function can issue PREPARE statement:
  - Benefit: opens up new functional possibilities for UDFs written in SQL PL

```
CREATE FUNCTION DYNSQLFUNC()
  RETURNS INTEGER
  VERSION V1
  DETERMINISTIC
  NO EXTERNAL ACTION
  PARAMETER CCSID UNICODE
BEGIN
  DECLARE VARCOUNT INTEGER;
  DECLARE LV_STMT_STR VARCHAR(256);
  DECLARE S1 STATEMENT;
  DECLARE C1 CURSOR FOR S1;
  SET LV_STMT_STR = 'SELECT COUNT(*) FROM
SYSIBM.SYSTABLES';
  PREPARE S1 FROM LV_STMT_STR;
  OPEN C1;
  FETCH C1 INTO VARCOUNT;
  CLOSE C1;
  RETURN VARCOUNT;
END!
```

SQL PL enhancements: support for constants

- Prior to Db2 12, a variable declared in a SQL PL compound statement could not be declared as a constant
- With Db2 12, user-defined constants can be declared in SQL routines and advanced triggers (available with function level V12R1M500)
- A few limitations:
  - Array-type variables cannot be declared as constants
  - SQL variables declared as constants are read-only
  - SQLCODE/SQLSTATE cannot be declared as constant SQL variables

```
…
DECLARE VAR2 INTEGER;
DECLARE cMAXVAL INTEGER CONSTANT 2000;
SELECT 1 INTO VAR2 FROM TEST WHERE VAR1 > cMAXVAL;
IF VAR1 > cMAXVAL THEN
  …
ELSE
  …
END IF;
```

SQL PL enhancements – source obfuscation

(available with function level V12R1M500)

- Historically, source for SQL PL routines (SQL procedures and UDFs) has been visible in SYSROUTINES in the catalog
  - Problematic for vendors (IBM included) that want to provide functionality in SQL procedures and/or UDFs while protecting intellectual property
  - Also problematic for users when SQL PL source is sensitive information
- Db2 12 provides ability to create and deploy SQL PL routines while “scrambling” the source in SYSROUTINES
  - Enabled via WRAP function, CREATE_WRAPPED stored procedure
  - Intellectual property in routine logic cannot be easily extracted
  - Applies as well to advanced triggers (source in SYSTRIGGERS)
  - **Note:**
    - Individual statements in SQL PL routine may be visible in SYSPACKSTMT
    - ALTER PROCEDURE and BIND DEPLOY not available for “wrapped” SQL PL routines (use DROP/CREATE as alternative to ALTER PROCEDURE)

MERGE before Db2 12: useful, but limited

```
MERGE INTO ACCOUNT AS A
USING (VALUES (:hv_id, :hv_amount)
FOR 3 ROWS)
AS T (ID, AMOUNT)
ON (A.ID = T.ID)
WHEN MATCHED
 THEN UPDATE SET BALANCE = A.BALANCE + T.AMOUNT
WHEN NOT MATCHED THEN INSERT (ID, BALANCE)
VALUES (T.ID, T.AMOUNT)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

Input can only be host variable arrays or a list of values

Only very simple “matched” and “not matched” clauses can be specified

- Only one update and one insert action can be specified
- Delete is not an option

Required when there are multiple “rows” of input values – the rows are processed separately, and processing continues if errors are encountered

AND, a target table row can be operated on multiple times in the execution of one MERGE statement – not always a desired behavior

Db2 12 enhanced MERGE: new capabilities

```
MERGE INTO RECORDS AR
USING (SELECT ACTIVITY, DESCRIPTION, DATE, LAST_MODIFIED
FROM ACTIVITIES_GROUPA) AC
ON (AR.ACTIVITY = AC.ACTIVITY) AND AR.GROUP = 'A'
WHEN MATCHED AND AC.DATE IS NULL THEN SIGNAL SQLSTATE '70001'
SET MESSAGE_TEXT = AC.ACTIVITY CONCAT ' CANNOT BE MODIFIED. REASON: DATE IS NOT KNOWN'
WHEN MATCHED AND AC.DATE < CURRENT DATE THEN DELETE
WHEN MATCHED AND AR.LAST_MODIFIED < AC.LAST_MODIFIED
THEN UPDATE SET
(DESCRIPTION, DATE, LAST_MODIFIED) = (AC.DESCRIPTION, AC.DATE, DEFAULT)
WHEN NOT MATCHED AND AC.DATE IS NULL THEN SIGNAL SQLSTATE '70002'
SET MESSAGE_TEXT =
AC.ACTIVITY CONCAT ' CANNOT BE INSERTED. REASON: DATE IS NOT KNOWN'
WHEN NOT MATCHED AND AC.DATE >= CURRENT DATE THEN
INSERT (GROUP, ACTIVITY, DESCRIPTION, DATE)
VALUES ('A', AC.ACTIVITY, AC.DESCRIPTION, AC.DATE)
ELSE IGNORE;
```

New input options: table, view, fullselect

Can use SIGNAL to provide customized error codes and messages

DELETE is now an option

Can have multiple “matched” and “not matched” clauses with various additional predicates, enabling multiple update, delete, and insert actions

New IGNORE option: when input row does not meet any “matched” or “not matched” conditions, ignore it

More on Db2 12 enhanced MERGE

- Available with function level V12R1M500
- New Db2 12 for z/OS MERGE capabilities mirror those provided by Db2 for Linux, UNIX, and Windows
  - Helps people who write SQL PL routines for Db2 for z/OS and LUW
- New Db2 12 MERGE capabilities available when NOT ATOMIC CONTINUE ON SQLEXCEPTION is NOT specified
  - In that case, in addition to new MERGE capabilities being available, there is different behavior:
    - A target table row can be operated on (via INSERT/UPDATE/DELETE) once
    - If error occurs during execution of MERGE, whole statement is rolled back
  - When input is in form of host variable arrays or list of values:
    - NOT ATOMIC CONTINUE ON SQLEXCEPTION is required
    - MERGE behavior is as it was prior to Db2 12

SQL pagination

- Background:
  - Common application requirement: present query result 1 page at a time
  - Example: return list of names from a directory, ordered by last name and then first name, in groups of 20 names, beginning with ERIKSON, MARY

Page 1

ERIKSON, MARY
ERWIN, PETER
EVANS, ROBERT
FAULK, LINDA
FEDERER, SUSAN
.
.
.
FIGGINS, SAMUEL

Page 2

FINNEY, MARY
FITZGERALD, ARTHUR
FOGLE, STEWART
FOLES, ANNE
FORREST, BRIAN
.
.
.
FRANKS, TERRY

…

Pre-Db2 12: the problem

1. Query predicate to generate result set is kind of convoluted:

```
WHERE (LASTNAME = 'ERIKSON' AND FIRSTNAME >= 'MARY') OR (LASTNAME >
'ERIKSON')
```

2. Getting subsequent pages of rows is not so simple:
   - Option: use data-dependent pagination
     - Get predicate values from last row in page n of result set and use them to get page n+1, using convoluted syntax above
     - Referring to the example on the preceding slide, that would mean:

```
WHERE (LASTNAME = 'FIGGINS' AND FIRSTNAME > 'SAMUEL')  OR (LASTNAME >
'FIGGINS')
```

   - Option: use ordinal position within result set as basis for pagination
     - Might do that with a scrollable cursor, or a SQL OLAP specification (e.g., ROW_NUMBER), or a SQL PL routine
   - These options are not very user-friendly from programmer's perspective

Db2 12 solution: enhanced SQL pagination (function level V12R1M500)

- Simpler predicate syntax for result set generation:

```
WHERE (LASTNAME, FIRSTNAME) >= ('ERIKSON', 'MARY')
```

  - Compare to syntax on preceding slide – simplification benefit increases with number of columns referenced in predicate (e.g., LASTNAME, FIRSTNAME, MIDDLE_NAME)
- Much simpler, more flexible way to present result set in pages
  - First page: `OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY`
  - Second page: `OFFSET 20 ROWS FETCH FIRST 20 ROWS ONLY`
  - Third page: `OFFSET 40 ROWS FETCH FIRST 20 ROWS ONLY`

New clause for queries – directs Db2 to skip over specified number of rows in result set before fetching

More new functionality

- Variable allowed in OFFSET clause (and FETCH FIRST clause)
  - Example: `OFFSET ? ROWS FETCH FIRST ? ROWS ONLY`…

"Piece-wise" DELETE

- Pre-Db2 12 problem: a DELETE such as the one below could affect a very large number of rows in a table

```
DELETE FROM T1 WHERE C1 > 7
```

  - Many locks could be acquired (or lock escalation could occur), and huge amount of data could written to log (backout would take a long time)
- Db2 12 solution: enable "piece-wise" DELETE (i.e., break large DELETE into smaller parts) via support for FETCH FIRST clause in a DELETE statement

```
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 5000 ROWS ONLY;
COMMIT;
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 5000 ROWS ONLY;
COMMIT;
```

Delete first chunk of rows

Delete 2nd chunk of rows

  - Available with function level V12R1M500

Arrays and global variables

- Global variables
  - Introduced with Db2 11, allow data values to be passed from one SQL statement to another without the use of application code
- Db2 arrays
  - Also introduced with Db2 11 – logically, like a “stack” of values
    - Two kinds: ordinary (array elements referenced by ordinal position in array) and associative (array elements referenced by associated index values)
  - With Db2 11, primary use of arrays is in SQL PL routines
    - Example: Db2 array can be an input parameter to a native SQL procedure
- Db2 12 (function level V12R1M500) provides several array and global variable enhancements

Db2 12 array and global variable enhancements

- Array-type global variables can be created
  - Extend Db2 array use cases beyond SQL PL routines
- ARRAY_AGG function (helpful for populating arrays) can be used with associative as well as ordinary arrays
  - Also, when using ARRAY_AGG to populate arrays with values from several columns of a table, ORDER BY only has to be specified once (versus once for each array)
- Global variables can have LOB data type (CLOB or BLOB)
- Global variable enhancements for SQL PL routines:
  - FETCH values into array global variables (for single-value global variable, FETCH… INTO does not have to be in a SQL PL routine)
  - Reference global variables in USING clause of EXECUTE statement, and in an OPEN CURSOR statement

In both cases, global variables provide substitution values for parameter markers

"Real" Unicode columns in EBCDIC tables

- **Db2 11 lets you have a Unicode column in EBCDIC table, <u>but</u>…**
  - Only for VARCHAR and VARGRAPHIC columns
  - Column internally represented as **VARBINARY** data type – results in several restrictions (on RI, VALIDPROCs and EDITPROCs, created and declared global temporary tables, indexes, …)
- **With Db2 12, you can have a regular byte-based Unicode column in an EBCDIC table**
  - Support for a wider variety of data types: VARCHAR, VARGRAPHIC, CHAR, CLOB, GRAPHIC, DBCLOB
  - **Real character, graphic and LOB data types** – removes most restrictions

Unicode in EBCDIC: Db2 11 to Db2 12

Db2 11

```
CREATE TABLE T1 (ID  INTEGER,
                 SURNAME  VARCHAR(128) CCSID 1208,
                 FIRSTNAME  VARCHAR(128),
                 BDATE DATE
                 ) CCSID EBCDIC;
```

SURNAME column represented internally as VARBINARY

V12

```
-ACTIVATE FUNCTION LEVEL (V12R1M500)
```

V12R1M500

```
ALTER TABLE T1 ADD COLUMN ADDRESS CHAR(50) CCSID 1208;
```

ADDRESS is a regular byte-based Unicode column

Migration of Db2 11 Unicode column

```
ALTER TABLE T1 ALTER COLUMN SURNAME SET DATA TYPE VARCHAR(128);
```

- **ALTER** can only be executed for **V11 Unicode columns**
- Same data type and data length
- CCSID clause is inherited from existing column

DSNTIJPM reports existing V11 Unicode columns

ODBC driver for z/OS enhancements

- Referring to ODBC driver used by local, z/OS-based applications
- Enhancement: support for KEEPDYNAMIC(YES)
  - Can improve performance for applications that repeatedly execute same dynamic SQL statement across commits
  - In Db2 12 system (function level V12R1M500 or later), when ODBC driver packages are bound with KEEPDYNAMIC(YES), two ways to preserve prepared statements across COMMITs:
    1. Set Db2 ODBC initialization keyword KEEPDYNAMIC to 1
    2. Set Db2 ODBC connection attribute SQL_ATTR_KEEP_DYNAMIC to 1
  - Result: driver does not re-prepare statement after COMMIT
    - With PREPAREs avoided, performance improves
- Another enhancement: Db2 ODBC driver now supports TIMESTAMP WITH TIME ZONE data type

New functions, newly optional correlation clause

- New built-in functions (function level V12R1M500)
  - Aggregate: PERCENTILE_CONT, PERCENTILE_DISC
    - Return percentile for set of values treated as points on a continuous distribution or as discrete values (e.g., "What salary is 90th percentile for department A01?")
  - Scalar: GENERATE_UNIQUE_BINARY, HASH_CRC32, HASH_MD5, HASH_SHA1, HASH_SHA256
    - HASH functions return hashed form of input – choice of 4 hashing algorithms
- Correlation clause now optional for table expressions*

From Db2 12 SQL Reference, for nested table expression (i.e. SELECT in FROM part of query):

```
►►──┬─TABLE─┬──(fullselect)──┬─correlation-clause─┬──────────────────────────►◄
```

This was formerly "on the line" in syntax diagram, meaning, "required"

  - Same goes for an XML table expression
  - With Db2 12, correlation clause for table expression required only if needed to refer to columns of table expression

* Retrofitted to Db2 11 via APAR PI55885

New built-in function: LISTAGG

"LIST," because it generates a list of values

"AGG," because it is an aggregate function

- Introduced with function level V12R1M501
- Syntax:

Removes duplicate values from list

What you want a list of

Values in list can be separated by commas, or by something else you specify

```
                .-ALL------.
>>-LISTAGG(-+----------+-string-expression-+---------------+-)--->
            '-DISTINCT-'                   '-,--separator-'

>--+-------------------------------------------------------+---><
   |                              .-,---------------------. |
   |                              V           .-ASC--.    | |
   '-WITHIN GROUP--(--ORDER BY----sort-key--+------+--)-+-'
                                             '-DESC-'
```

Optional clause - lets you specify order of values in list

LISTAGG in action

- Given this data:

EMPLOYEE table

EMPNO	LASTNAME	WORKDEPT
0001	THOMAS	A01
0002	ROGERS	B01
0003	HONG	A01
0004	BARKER	B01
0005	KOHL	B01

- This statement:

```
SELECT WORKDEPT, LISTAGG(LASTNAME, ', ') WITHIN GROUP(ORDER BY LASTNAME)
 AS EMPLOYEES
 FROM EMPLOYEE
 GROUP BY WORKDEPT;
```

- Yields this result:

```
WORKDEPT EMPLOYEES
-------- --------------------
A01      HONG, THOMAS
B01      BARKER, KOHL, ROGERS
```

Comma-separated list of employees, by department, in ascending last-name order within department

LISTAGG DISTINCT (to remove duplicate values)

- Given this data:

ORDER_NUM	CUSTOMER	ORDER_DATE
0001	COX INDUSTRIES	2017-06-15
0002	ACME	2017-06-20
0003	COX INDUSTRIES	2017-06-22
0004	BILCO	2017-07-03
0005	ACME	2017-07-09
0006	BILCO	2017-07-18

ORDER table

- This statement:

```
SELECT MONTH(ORDER_DATE) AS MONTH,
 LISTAGG(DISTINCT CUSTOMER, ', ') WITHIN GROUP(ORDER BY CUSTOMER)
 AS CUSTOMERS
 FROM ORDER
 GROUP BY MONTH(ORDER_DATE);
```

- Yields this result:

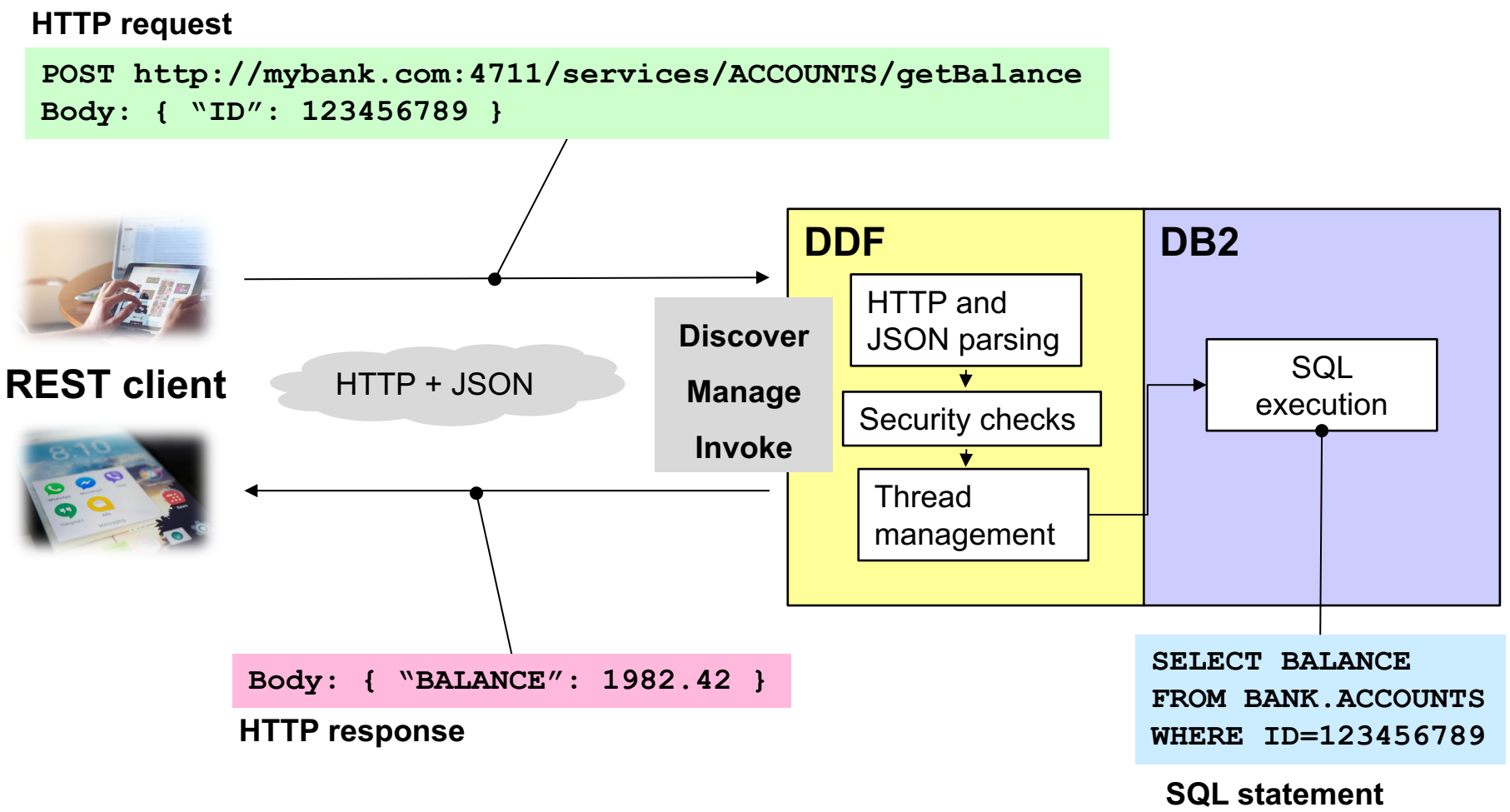
```
MONTH       CUSTOMERS
-------------------------------
    6       ACME, COX INDUSTRIES
    7       ACME, BILCO
```

Comma-separated list of customers that placed orders with us, in ascending customer-name order within month of order placement

Db2's native REST interface

- Introduced with Db2 12, retrofitted to Db2 11 via APARs PI66828, PI70477
- An extension of Db2 distributed data facility (DDF) functionality
  - Leverages DDF capabilities including thread pooling, classification, accounting/statistics tracing
  - Leverages existing Db2 package management capabilities
  - SQL statements executed via REST calls run under preemptible SRBs in the DDF address space
    - SQL executing under DDF preemptible SRBs is up to 60% zIIP-eligible
  - A single static SQL statement can be exposed for execution via a REST call
    - Could be a single data manipulation SQL statement (SELECT, INSERT, UPDATE, DELETE)
    - Could be a call to a Db2 stored procedure (in that case, use native SQL procedure if possible, to get zIIP offload – thanks to running under preemptible SRB in Db2 DDF address space)
- Designed for high performance
  - IBM tests: 540 million transactions per hour through the Db2 for z/OS REST API

Db2 RESTful services in action



Thanks for your time.

Robert Catterall, IBM
rfcatter@us.ibm.com